**Task -1:-** **Write a C program to reverse a given string without using any additional library functions.**

#include <stdio.h>

void revStr(char\* str)

{

int length = 0;

int i;

char temp;

for (length = 0; str[length] != '\0'; length++);

for (i = 0; i < length / 2; i++)

{

temp = str[i];

str[i] = str[length - i - 1];

str[length - i - 1] = temp;

}

}

int main()

{

char str[100];

printf("Enter a string: ");

scanf("%s", str);

revStr(str);

printf("Reversed string: %s\n", str);

return 0;

       }

**Task-3**:- **Explain the concept of structures in C and write a program to store student information (name, roll number, marks) using a structure.**

In C, a structure is a user-defined data type that allows the combination of data items of different kinds. Structures are used to group together different types of variables under a single name. Each variable within a structure is called a member.

The syntax to define a structure is:

struct structure\_name {

data\_type member1;

data\_type member2;

...

};

#include <stdio.h>

struct Student {

char name[50];

int rollNumber;

float marks;

};

int main() {

struct Student student;

printf("Enter student name: ");

scanf("%s", student.name);

printf("Enter roll number: ");

scanf("%d", &student.rollNumber);

printf("Enter marks: ");

scanf("%f", &student.marks);

printf("\nStudent Information:\n");

printf("Name: %s\n", student.name);

printf("Roll Number: %d\n", student.rollNumber);

printf("Marks: %.2f\n", student.marks);

    return 0;

}**Task-4:- Differentiate between single-linked lists and doubly-linked lists in C. Write code snippets to create a node and perform a basic insertion operation in a singly-linked list.**

Differences Between Singly-Linked Lists and Doubly-Linked Lists in C

|  |  |
| --- | --- |
| **Singly-linked Lists** | **Doubly-linked Lists** |
| Each node contains data and a pointer to the next node. | Each node contains data, a pointer to the next node, and a pointer to the previous node. |
| Navigation is only possible in one direction (from the head to the tail). | Navigation is possible in both directions (forward and backward). |
| Less memory usage compared to doubly-linked lists because each node has only one pointer. | More memory usage because each node has two pointers. |
| Insertion and deletion operations are simpler and slightly faster since only one pointer needs to be updated. | Insertion and deletion operations are more complex as both next and previous pointers need to be updated. |

#include <stdio.h>

#include <stdlib.h>

struct Node {

int data;

struct Node\* next;

};

struct Node\* createNode(int data) {

struct Node\* newNode = (struct Node\*)malloc(sizeof(struct Node));

if (!newNode) {

printf("Memory allocation error\n");

exit(1);

}

newNode->data = data;

newNode->next = NULL;

return newNode;

}

**Task-5**:- **Explain the concept of pointers in C and write a program to swap the values of two variables using pointers.**

A pointer is a variable that stores the memory address of another variable. Pointers provide a powerful and flexible way to manipulate data and memory, allowing for dynamic memory allocation, efficient array handling, and the creation of complex data structures like linked lists and trees.

**Key Concepts:**

1. **Declaration**: A pointer is declared using the asterisk (\*) symbol. For example, int \*ptr; declares a pointer to an integer.
2. **Address-of Operator (&)**: The address-of operator is used to get the memory address of a variable. For example, ptr = &var; assigns the address of var to the pointer ptr.
3. **Dereference Operator (\*)**: The dereference operator is used to access the value stored at the memory address pointed to by the pointer. For example, \*ptr gives the value of the variable that ptr points to.
4. **Pointer Arithmetic**: Pointers can be incremented or decremented to point to the next or previous memory location.

#include <stdio.h>

void swap(int \*a, int \*b) {

int temp;

temp = \*a;

\*a = \*b;

\*b = temp;

}

int main() {

int x, y;

printf("Enter value for x: ");

scanf("%d", &x);

printf("Enter value for y: ");

scanf("%d", &y);

printf("Before swapping: x = %d, y = %d\n", x, y);

swap(&x, &y);

printf("After swapping: x = %d, y = %d\n", x, y);return 0;

}